24P

BOLT BERANEK AND NEWMAN INC

CONSULTING • DEVELOPMENT • RESEARCH

BBN Report No. 2792
A.I. Report No. 1Ø

"ROBOT" COMPUTER PROBLEM SOLVING SYSTEM

Quarterly Progress Report

Contract No. NASW-2572

Joseph D. Becker
E. William Merriam

23 January 1974

Prepared For:

National Aeronautics and Space Administration
Washington, D.C. 2Ø546

BBN Report No. 2792
A.I. Report No. 10

Bolt Beranek and Newman Inc.

"ROBOT" COMPUTER PROBLEM SOLVING SYSTEM

Quarterly Progress Report

Contract No. NASW-2572

Joseph D. Becker
E. William Merriam

23 January 1974

Prepared For:

National Aeronautics and Space Administration
Washington, D.C. 20546

TABLE OF CONTENTS

INTRODUCTION

The following is a report on progress made on NASA Contract Number NASW-2572 for the period 23 October 1973 to 23 January 1974. This work is a continuation of work started under NASA Contract NASW-2236 and previously reported in BBN Report Numbers 2316 and 2646.

Our work on this contract encompasses the conceptual, experimental, and practical phases of the development of a Robot Computer Problem Solving System. We categorize the progress made this quarter in the conceptual and practical domains as follows:

I. Conceptual Issues
   Clarification of the nature of robot intelligence

II. Practical Issues
   A. Continued progress in converting the programming language SAIL to run under the TENEX monitor

   B. Investigation and specification of the issues involved in the practical use of the ARPA network for running several cooperating jobs at different host sites

   C. Assistance with the JPL Robot Project in connecting JPL facilities to the ARPA network, in running demonstrations, and in using the BBN TENEX facility

These topics are discussed in the following pages, using the same outline as that given above. Included with each topic is an indication of our plans for future work. As no work was done this quarter in the experimental domain, that aspect will not be discussed in this report.

I. CONCEPTUAL ISSUES

During this quarter we have taken some time off from direct work on our robot simulation (the experimental portion of our work) in order to clarify the nature of the "robot intelligence" that we are trying to develop. This has involved us in considerable discussion of the nature of intelligence in animals, since animals exhibit many sorts of behavior that one might expect from intelligent, roving robots. We have tried to focus on the properties of observable animal behavior, rather than postulating "intellectual" mechanisms derived from an analysis of human thinking. In particular, we have concentrated on the notion that sensori-motor skills may account for a good deal of what we perceive as "intelligence" in the behavior of animals and roving machines.

A. Some Characteristics of "Intelligence" in Animals

It goes without saying that what we call "intelligence" in animals is not the ability to think great thoughts or score well on IQ tests, but rather has something to do with the quality of the animal's behavior in its struggle to stay alive. Thus, cats are more intelligent than frogs, and frogs are more intelligent than worms.

With regard to activities which an animal initiates of

its own accord, such activities tend to be considered intelligent insofar as they have a recognizable goal. For example, when a cat stands on his hind legs and turns a doorknob with his forepaws, we recognize the objective, and therefore consider his action intelligent. When the cat roams restlessly around the room, we do not readily perceive his objective, and so we do not feel that this behavior is especially intelligent. Yet, we should realize that this exploratory behavior probably has some latent survival value. Indeed, in many animals goal-directed behavior occupies only a limited percentage of their waking hours; the bulk of their time goes into apparently non-directed activities such as exploration, play, or just sitting around. It may not be wise to exclude non-directed activities from the investigation of animal intelligence simply because we do not understand how they relate to the behaviors that are more recognizably "intelligent".

With regard to activities in which an animal responds to events occuring in its environment, such activities tend to be considered intelligent insofar as the response is fast and appropriate. For example, many animals (including humans) will follow you with their eyes as you walk in front of them. When a movement-detecting electric-eye apparatus or a painted portrait seems to do the same thing, you will have the uncanny feeling of being watched, i.e. kept under observation by some intelligence, even though you know these

objects to be inanimate. Similarly, the immediate and appropriate responses of the Venus's-flytrap give us the eerie feeling that the plant knows what it is trying to do. Even reflexes, then, give the appearance of intelligence because they constitute fast, appropriate reactions.

The "appropriateness" of a response is measured partly by its adaptiveness to the particular stimulus event. Thus, a dog who is good at catching dog-biscuits seems less intelligent if we discover that he will still snap his jaws shut even when we don't actually throw the biscuit, but merely feign a throwing motion.

Finally, an action, either freely initiated or responding to the world, seems more intelligent insofar as its effect is removed from the animal's (presumed) ultimate goal. For example, the cat turning a doorknob with its paws is presumably trying to turn the knob so as to open the door, and this in order to enter the kitchen, and this in order to check out the food bowl. By contrast, the steps in the performance of an integrated motor activity, such as the footwork involved in a difficult jump by a cat, do not appear to be intelligent, even though their actual logic may be very much more subtle than the logic of turning a knob to open a door. In a sense, the jump appears to be a less intelligent act precisely because it is done more proficiently: we are not able to analyze it into a discrete

sequence of subgoals, so its component actions appear to contribute directly to the main goal.

To summarize, we have said that some of the characteristics of "intelligence" in animal activity are goal-directedness, separation of the action from the goal, and the speed, appropriateness, and adaptiveness of a response. In all of this discussion we have carefully spoken of activities seeming or being considered to be intelligent -- for it seems clear to us that "intelligence" is an attributed rather than an inherent property of behavior. We have already seen a number of examples where our perception of intelligence can be quite misleading with respect to a total analysis of an animal's behavior: for example, we disregard activities such as play because we do not know their purpose, we write off the process of coordinating a jump since we perceive it as a single action leading directly to its goal, and we may even attribute some aspects of intelligence to electric-eye systems, plants, and portraits.

The fact that "intelligence" is an attributed characteristic -- and one which does not adequately cover the properties of animal behavior that we might be interested in employing in machines -- is very important for our work in developing a simulation of an "intelligent" robot. In particular, it means that people (including us)

will tend to judge the merits of the simulation on the basis of its final behavior, rather than on the basis of its mechanism, even though we are explicitly interested in uncovering the mechanisms of intelligence and not in putting on an impressive show. To put it another way, a simulation model that appeared impossibly stupid on one computer might appear quite "intelligent" when run on a computer that was 50 times faster, simply because rapidity of response is one of the subjective indicators of intelligence. This means that we must be extremely cautious in interpreting the results of our simulations, and that the behavior of the simulation is not a simple indicator of the merits of the ideas on which the simulation is based.

B. The Motivations behind Intelligent Behavior in Animals

One of the most difficult problems in simulating a cognitive system is to provide the simulation with any motivation for thinking at all! An animal, when it is born, sets about its business without being told what to do; but the world's most "intelligent" computer system, when turned on, will do nothing at all until it is given explicit motivating instructions. Because this problem weighs heavily on our robot simulation, we have given some thought to the kinds of motivation that animals can be said to experience. We can distinguish at least four categories.

Drives: The notion of a "drive" has been used to account for much of animals' freely-initiated behavior. Usually, one simply postulates some measurable or at least parametric quantity (hunger, drowsiness, lust, etc.), some factors which affect the variable (e.g. activity increases hunger and drowsiness, lust may have a monthly cycle, all three perhaps increase with the passage of time, etc.), and some sort of detector which spurs the animal into an appropriate activity when one of these quantities becomes more (or less) than the animal can bear. It is a form of model that is simple, nearly tautological, and indispensable.

Responses: We hardly need to mention that the ideas of "response" and "reflex" have played far too large a role in the traditional description of animal behavior, but that they of course do represent important components of animal activity. There is no obvious way of defining what a "response" is, or even a "direct response" to an external event. For example, consider the tracking of a moving object that we mentioned earlier; it is performed by most higher animals at rest, including humans, when a moving object enters their field of view. Now, it is not at all an elementary process from the sensori-motor point of view, as we have learned from our own simulations of tracking behavior. Yet, it is evoked in a manner that does seem "reflexive", namely in an apparently automatic, non-directed

fashion. There may be no easy way to define the point at which "reflexive" response leaves off, and intentional response takes over.

Goals: As we mentioned in the previous section, many of the behaviors that we call intelligent in animals are those to which we can assign apparent goals or results. When an animal stalks a prey that it has seen, frees itself from a trap, or finds the warmest place around to sleep in, we feel that we understand the structure of the behavior because we understand the end toward which the behavior is directed. Interestingly, the ultimate motivation behind most goal-oriented behavior in animals is describable by a simple drive (toward food, warmth, etc.). This means that a concept which is vital in describing the behavior of men (and of robots), namely the concept of constructive work, is nearly irrelevant in the world of animals. That is, a man will commonly be given (or give himself) a task such as making a chair or learning to play the guitar, where the basic motivation is very far from any obvious survival drive. Similarly, we will certainly expect any robots that appear on the scene to work for their living. But animals by-and-large lead unconstructive lives, and they give us only a few examples (such as beavers) of large-scale activities whose end-result is not the simple satiation of a drive (even if we are generous enough not to attribute beavers with a simple "dam-building" drive). The notion of

constructive work, then, calls for motivational models beyond those which have arisen from the study of animal behavior.

"Other": Although most animals do not work, many of them definitely play, and we have already seen that play belongs to a category of non-(apparently)-directed activities whose purpose is so unobvious that we might wish to forget about them altogether. Unfortunately, this class contains types of activity that are vital to many robot applications, such as "exploration", "curiosity", and "vigilance". And although no one would intentionally build a robot that had to spend a large fraction of its time sleeping, we have to admit that the exclusion of sleep is a totally uninformed decision, since we have no idea of its (apparently indispensable) function in living systems. Then too, the many facets of "learning" all fall under the category of activities which are "non-directed", in that they do not bring an immediate reward; yet they are absolutely vital to any system that is to make the barest pretenses to intelligence. In a word, we have almost no idea of how to model the motivational basis of activities in this category, since we only poorly understand their function. Perhaps considerable progress in the understanding of intelligence can be made by further investigation of this much-neglected area.

To summarize the discussion of the motivations that underlie intelligence, we may ask why an organism tries to be intelligent at all, or indeed why it tries to do anything at all. The answer is that there is no necessity: the majority of living organisms, being plants or plant-like, "do" almost nothing in the way of real activity. A minority of organisms have found that they can live better by actively affecting their environments, but such improvement comes at the expense of greater complexity, and therefore a greater investment in the safety of the individual. (That is, plant species do not mourn the loss of billions of seeds each season; ant colonies freely sacrifice hundreds of individuals; but the higher animals place an increasing reliance on the survival of a fairly high percentage of the individuals that are born.) Intelligence, then, is a parameter which describes the degree to which an organism can adapt its environment to suit itself (for its own betterment, e.g. it can go out and hunt food instead of waiting for it to come by) and at the same time it describes the degree to which it can adapt itself to suit the environment (for its own survival, e.g. when it cannot find food it may have to migrate). Because of this activity-safety trade-off, the lives of the more intelligent species are not "better" or even more "efficient" in any sense than the lives of less adaptive organisms -- the former are merely more exciting.

C. The Basis of Intelligent Behavior in Sensori-motor Skills

Putting aside now the question of motivation, let us suppose that an animal or robot is in a given situation, and needs or wants to do a certain thing or obtain a certain result. What constitutes intelligent action? As we saw in the first section, it is, in part, action which is quickly produced, which is appropriate to the situation and the desired result (i.e. effective in bringing about the result). Now, nothing we have said so far precludes the possibility of the system's action being "canned", i.e. memorized (or built-in) in advance, rather than being extemporized on the spot. To put the matter baldly, we can imagine a system which can discriminate only a relatively small number of situations S1,S2,...Sn (where a "situation" includes both perceived external conditions and given internal motivations), and which is somehow already programmed with the optimal responses R1,R2,...Rn corresponding to each situation.

Within its limits, such a system is by definition optimized and above cavil. Indeed, its quick, appropriate responses may well impress an observer with their "intelligence" ... until he discovers the element that is missing: adaptability. But it is important to repeat that intelligence (or adaptability) is not a necessity to life, but rather an optional commodity which is present in

different species to different degrees. Many lower animals lead pleasant lives that are susceptible to the simple sort of description just given. Moreover, there is every reason to believe that species which do employ adaptive mechanisms combine them with a system of "canned" responses, rather than simply replacing the canned responses with fancier means of deciding what to do.

This assertion, that "intelligent" systems maintain a large reliance on a canned-response mechanism, is amply supported by observations of instinctual behavior and "innate releasing mechanisms" in higher organisms. It is also supported by properties of skilled behavior in humans. A human can stumble through any number of intricate behaviors (playing a piano piece, weaving a reed basket, driving a car, etc.), but he cannot perform any such activity at all well until he has practiced it so often that it comes "automatically" -- that is, until he is "skilled" at it -- that is, until he has "canned" the behavior. For this reason, we have identified the concept of a "canned response" with the more dignified term "sensori-motor skill". The use of the latter term also guards against an interpretation of our discussion in terms of "stimulus-response" psychology, which addresses the same issues with much the same perspective, but at such a level of oversimplification as to be totally useless.

The problem with simply dividing all behavior into a finite number of possible situations and a corresponding number of skilled responses is that the world presents an organism with a boundless variety of situations rather than just a few. The more discriminating the organism's sensory systems are, the more work its cognitive system will have to do in order to un-discriminate (i.e. recognize) what situation faces the organism and what action should be taken. This extra cognitive activity amounts to the "intelligent" use of the otherwise primitive mechanism of pre-programmed skills. It has at least three major aspects:

Generalization, or adaptiveness: At the simplest level, "generalization" refers to the process of recognizing that two situations are equivalent or similar with regard to the response that they demand from the organism. "Adaptation" refers to the other end of the same process, wherein the organism tailors its response -- that is, modifies the program that consitutes its skilled behavior -- so as to suit the new variant of a previously-encountered situation. It should be noted that fully-developed skills are, unlike simple reflexes, open to extreme adaptation; for example, the skill of "hitting a backhand" in tennis applies to an infinite variety of initial conditions, and to a fair variety of desired results, all imposed on the same basic set of actions. What does it mean for a skill to be "basically" well-defined, and at the same time open to

infinite extemporaneous variation? This is one of the major questions in capturing the essence of intelligent behavior.

Coping with the volume of experience: The aspect of generalization which is simplest to state but hardest to explain involves the sheer size of the past history that any higher organism has at its disposal for possible generalization. A human being, for example, can be shown an unusual object (such as a boomerang) and will often recognize it in a matter of seconds, even though he has not seen one for many years, and then only as a picture in a book. Indeed our recognition of commonplace, standardized objects (e.g. telephones) is no less remarkable, since the potential search space of experience is presumably just as large -- namely all the experience of a lifetime. Even when all the other many difficulties of the "recognition" process are put aside, the question of how we apparently search through several decades of experience in a second or two is one of the greatest unsolved enigmas in the functioning of the brain, and one of the greatest unsolved problems in the creation of computer models of intelligent behavior.

Coordination: An aspect of skilled behavior whose importance is just becoming clear to us is that of the temporal coordination of responses. Coordination becomes an issue whenever a situation can be factored into two or more recognizable sub-situations. For a system of any

intelligence, this is almost always the case, since there are usually several perceived objects in the environment, which constitute "sub-situations". The problem, of course, is how to respond to several things at once. Here are a few typical examples from the life of an animal or a robot:

(a) the organism is sitting still, and wishes to look at two nearby objects, but one is to the left and one is to the right, so that it cannot look at both of them at once;

(b) the organism is moving along a familiar path when it suddenly notices a previously unseen object lying near the path;

(c) the organism is moving toward an object in order to pick it up, which involves simultaneously moving and keeping its eye on the object, then slowing down and stopping near the object on the basis of visual judgements of distance from the object.

In the first example, two external sub-situations are competing for attention; in the second example, a new situation interrupts an ongoing activity; in the third example, an intentional activity (going to pick up an object) has two separate sub-activities (moving toward the object and visually perceiving it) which are vitally dependent upon each other. No simple "stimulus-response"

notion of the production of behavior can come close to explaining these intricate coordinations which characterize animal or robot behavior in even the simplest of tasks. What is needed is a full-scale investigation and model of what constitutes a coordinated skill.

We feel that the animals that we consider intelligent possess a great number of highly-refined sensori-motor skills, and the totality of these skills may account for a substantial portion of what we regard as "intelligence" in these animals. At any rate, flexible but pre-programmed units of behavior certainly form the substrate on which any more adaptive intellect must be built. Therefore, we foresee that the next step in our research will be the formulation of a precise definition for the notion of "sensori-motor skill" that will take into account generalization, coordination, and other such problems which arise in the behavior of both animals and robots. Our main tool in formulating this definition will be our robot simulation, which will allow us to test and refine our ideas as rapidly as we can propose them. Then, once a workable definition of "skill" is arrived at, the robot will undoubtedly lead us directly into the study of the acquisition and employment of skills, and their relationship to intelligent behavior.

II. PRACTICAL ISSUES

For the past year, we have been providing assistance to the JPL Robot Project in several miscellaneous areas such as: the use of the ARPA network; configuring the JPL IMLAC computer as well as developing software for it; Machine Intelligence research; developing and running demonstrations.

During this quarter we continued to provide this sort of assistance, the general nature of which is outlined in section C below. However, based on a year's experience and on the feeling that some of the requested assistance (which included requests from many more sources than we could hope to satisfy) would not truly benefit the project, we decided to attempt to get a clear overall understanding of the project before fulfilling any particular request. (There were exceptions to this where the need was immediate and clear -- see section C below.) Also bearing on this decision is our feeling that our knowledge of system-building and Machine Intelligence can benefit the JPL robot project far more than our merely helping out with almost incidental items as we are now doing. Thus we have gathered together our various notes, started to discuss the details of the project with each member of the JPL Robot Project team (by telephone), and have planned a trip to JPL early next quarter to continue these discussions in person.

Our hope is that as a result of these discussions we will be able to define those areas in which we can best assist the project. The structure of the remainder of this report and our recent work reflects the change in our involvement with the JPL project in the following ways:

1) We are attempting to quickly complete the work on converting the SAIL language to run on the TENEX computing system, with the result that this effort will no longer divert our attention from more important issues of robot system development (section A).

2) We are exploring (in some depth) issues which are important to the JPL project: investigating and specifying the issues involved in the practical use of the ARPA network for running several cooperating jobs at different host sites (section B).

3) We are continuing to provide miscellaneous support as before, but will report it here and in subsequent reports only in summary form (section C).

A. Conversion of SAIL to TENEX

During this quarter, we finished Version 1 of BBN-TENEX SAIL and made it available to JPL and the rest of the SAIL community. Preliminary documentation was made available via a file on the BBN-TENEX system.

In this initial system both the segment and the compiler have been completely TENEXized. This means that they do not use either the TENEX PA1050 (compatibility) program or DEC I/O instructions and monitor calls (UUO's). Instead JSYS's, the TENEX versions of these monitor calls, are used. Additionally, file names in TENEX format (an extension of the standard DEC format) are recognized and the TENEX "command completion" feature is implemented for reading file names for compilation. (This last feature necessitated the implementation of a completely new command scanner in the compiler.) Finally, a package of ARPANET utility functions has been incorporated into the system. These functions are based on the XNTLIB package developed by the BBN-TENEX Distributed Computation Group.

Next quarter, we expect to deliver Version 2 of BBN-TENEX SAIL. In it we hope to have some known bugs fixed as well as to provide a new interrupt facility which is needed by JPL.

B. Investigation into the Possible Use of the ARPANET for a Multi-Host Robot Program

1. Background

During this quarter we began to investigate the possibility of using the ARPA network to run several interacting programs simultaneously at different host sites. By running concurrently and interacting with one another, the programs would form a complete system to do a given task. Ideally, we would like the different program components to be able to run on different types of machines on the ARPANET, taking advantage of machines with lighter load factor or greater efficiency (e.g. a machine with a highly efficient FORTRAN system could be used for a physical world simulator). In addition, such a multi-host program could be designed to continue running, even if some hosts became temporarily unavailable.

Running several interacting programs concurrently on different machines of the same type is a difficult enough problem without introducing the vagaries of a method that would allow programs to run simultaneously on several types of machines. Thus, in this discussion we assume that the programs will be run on many different physical machines, but only on one type of machine -- namely, one using a TENEX operating system.

Although the ARPANET has been available for several years, there has not yet been a great deal of investigation of the problems of running mutually interacting programs on several hosts within the network. The two major examples of multi-host programs are the RSEXEC program which is an ongoing attempt to create a unified operating system and environment for the TENEX systems on the network, and McRoss, a multi-computer air-traffic control simulator (also operating on TENEX systems). Both of these have provided experience in the techniques of organizing multi-host programs, but neither provides the basis for something like a multi-host robot program. In particular, neither provides a system capable of taking the description of a multi-host program, initiating it, maintaining control over it, and providing the user with facilities for debugging the program and communicating with its various components.

If the processes into which the multi-host program would be split were all written in one programming environment (e.g. INTERLISP), or indeed if it were certain that the decomposition of the program into independent processes would not change over time, one might be tempted to develop a limited, special-purpose system capable only of initiating and controlling a particular multi-host program. Since neither of these conditions holds for our robot simulation nor for JPL's robot project, we have instead begun to investigate and design the tools needed to work

with a more general class of multi-host programs. We are not aiming for "full generality" (indeed, there is so little experience in this area that it is unclear what would constitute full generality), but instead are trying to develop a system which will enable us to work conveniently with a multi-host program as it evolves, and will still provide a reasonable degree of efficiency.

2. Our Current Work on Multi-host Program Facilities

During this quarter we have proceeded on four parallel tracks in the development of a multi-host version of our robot. We have redesigned our current robot simulation, splitting it into three independent processes. We have investigated the facilities currently available for implementing multi-host programs on the TENEX computers of the network, including both the facilities available in TENEX and programs written by individuals at BBN and XEROX PARC. This has involved writing several small-scale experimentation programs, including a LISP program which can initiate jobs at several network TENEX sites. We have investigated the additions to TENEX planned as part of the BBN "Distributed TENEX" effort to determine which of them could be used in a multi-host system. Finally, we have tried to produce a preliminary description of the features we would like to have in a facility for developing and

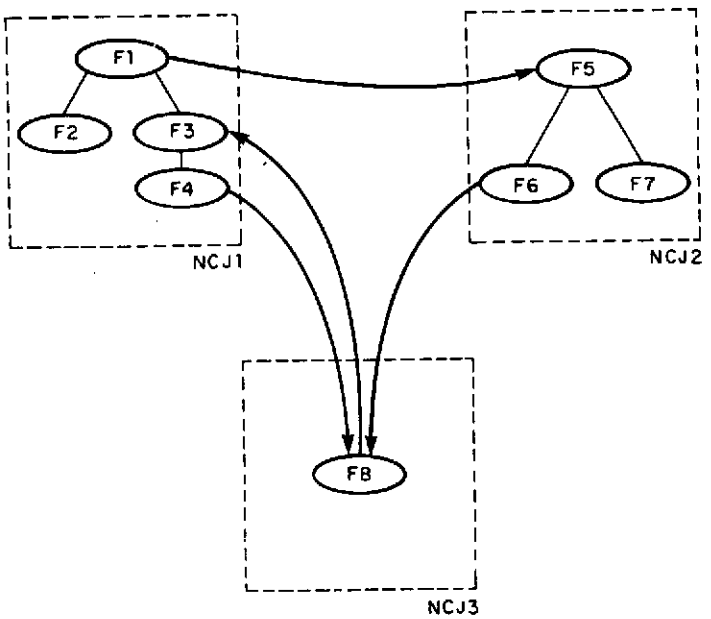
running a multi-host robot.

3. General Description of the Operation and Structure of a Multi-host Program

Considering only the computations to be performed by a multi-host program, such a program might be described as a network whose nodes were jobs residing on (possibly distinct) TENEX hosts on the ARPANET. Each job would perform part of the computation of the multi-host program and would consist of one or more forks within the given TENEX host. The arcs of the network would be the communications paths (possibly ARPANET connections) between the jobs on the different hosts.

Figure 1 shows a conception of such a multi-host program with three nodes. The dashed lines delimit the nodes of the network, the heavy solid lines represent the arcs or data paths, with arrowheads indicating the direction of information flow. Each node is a job whose fork structure is indicated by ovals representing forks (with arbitrary names F1, F2, etc.) and thin lines representing the relation between a fork and its immediate inferior forks. In the remainder of this discussion we will refer to such jobs in a computational network as Node Computational Jobs or NCJ's.

Figure 1

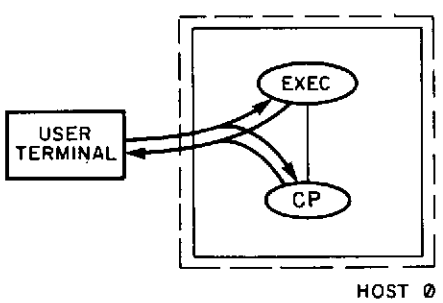


Once such a multi-host program was debugged and running, with NCJ's on various TENEX sites in the ARPANET, it would be conceivable for the user to communicate with and control the NCJ's by using the TELNET program available in TENEX. However, distributing the NCJ's to different sites, establishing connections, and controlling and communicating with the NCJ's during debugging are non-trivial procedures. Things would be simplified by the existence of a "computational network control program" (CNCP), consisting of a network of jobs (each called a "controlling node" or CN) on various TENEX sites, coordinated by a central "control program" or CP. Thus, in our view, a complete multi-host computational system would consist of a computational network (such as that shown in Figure 1)

established, controlled and serviced by a Computational Network Control Program.

The following sketch of a scenario for building and operating a multi-host program will provide the framework for later discussions of our ideas on the structure of a multi-host system. The user logs on to some TENEX computer on the ARPANET (possibly through a TIP) and then starts up a Control Program (CP). The CP will provide him with control over the Computational Network Control Program (CNCP), the multi-host equivalent of loader, debugging system, and TELNET-like communications facility. (See Figure 2.) Then, either by typing in directly, or more probably by specifying

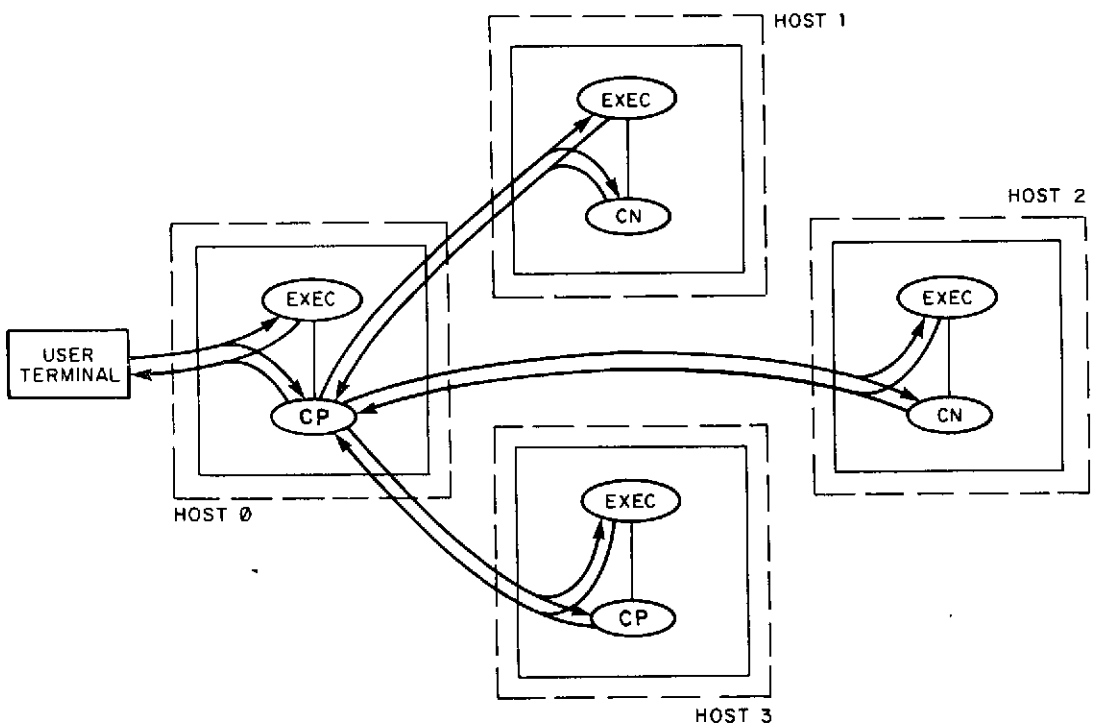
Figure 2



a set of files, he will provide the CP with a description of the multi-host program that he wishes to establish. This description will contain such information as the fork structure of the various NCJ's to be run, the communications channels to be established between the NCJ's, and the

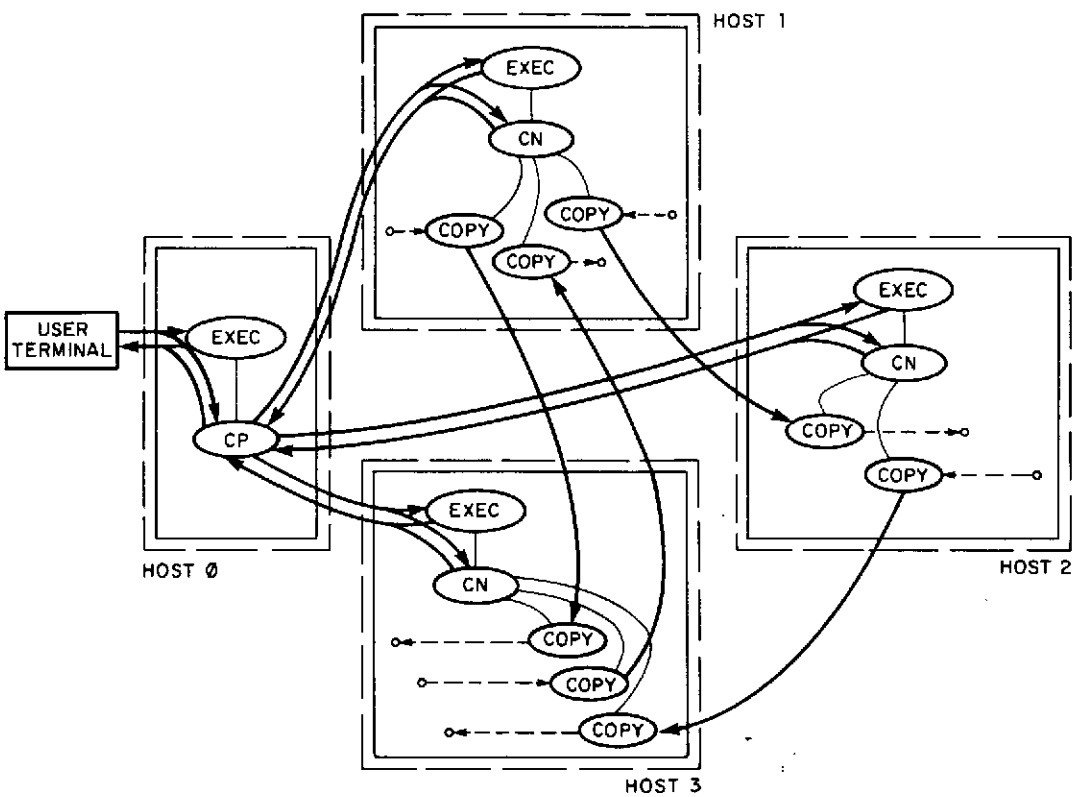
facilities (hardware, disk space, etc.) needed by each NCJ. The Control Program will then determine which TENEX sites are suitable for the various jobs (taking into account such factors as machine load, charge rate, etc.), and will log in (with the user's name, password, and appropriate account number) at the determined sites. At each site it will start up a copy of the Control Node (CN) program and set up the communications and control paths to each node. (This is shown in Figure 3.) It will then establish the ARPANET connections and the copying forks (denoted by COPY in the

Figure 3



diagrams) needed for the inter-NCJ communications channels.(*) (See Figure 4.) Each CN will then load the corresponding NCJ from its files (on some TENEX system or systems), setting it up as a subordinate fork structure, and complete the inter-NCJ channels. (See Figure 5.) All of this will proceed without the user performing any action beyond indicating the desired network structure.

Figure 4

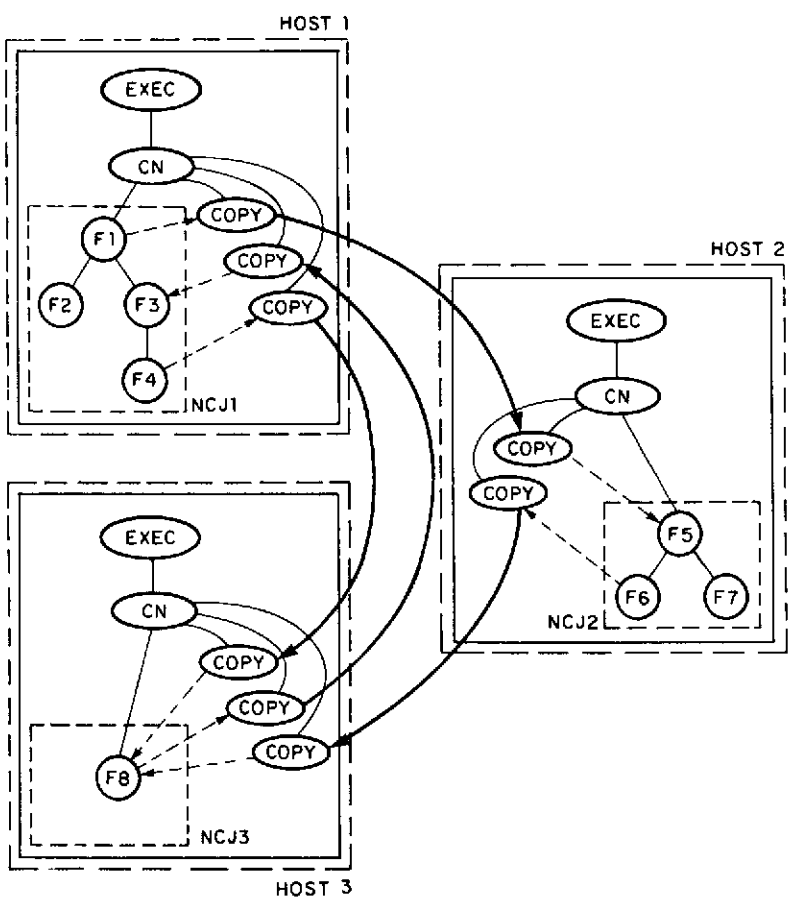


---

* The details of our proposed implementation of inter-NCJ channels are given below.

On the user's command, the Control Program will cause the CN's to start up one or more of the NCJ's in the network. As the computation progresses the Control Program will allow the user to conveniently direct input to any desired job, to queue or receive output as desired from any NCJ, and to monitor the inter-NCJ communications channels. In addition, the Control Program will have various debugging facilities. These will include such single fork debuggers as DDT or the INTERLISP debugging package for each NCJ, and mechanisms to permit the user to suspend the operation of

Figure 5



one or more NCJ's depending on the information flowing in the communications channels or the state (e.g. breakpoint) of some single job.(*)

We describe below some of the ideas we have developed for implementing such a multi-host computational network facility. These ideas are not to be construed as a final description of the system we would like to see available. They do provide a framework for describing such a system, but in fact raise more questions than they answer. In addition, the structure of the system described is based on the existence of several TENEX facilities which are not yet available, as indicated below.

When we started this investigation we discussed our ideas with members of the TENEX development group at BBN. They showed us their proposals for several new features in the TENEX system which they thought might make our work easier. The first feature is a system for trapping JSYS calls to the monitor, to allow (among other things) extension of user-level I/O facilities and the provision of a "distributed file system" at the user level. At present, the RSEXEC system allows the user to act as if the files he

---

* It would be desirable for the Control Program to monitor the state of the various sites and network links and to restart the computational network with minimal loss of time and information if a site or connection went down. This may, however, require capabilities not yet available at the individual TENEX sites.

has at a number of TENEX sites are available as a single file structure, without his being concerned where they are located. A JSYS level distributed file system, implemented with the JSYS trap mechanism, would enable the user's programs to operate in an equivalent environment in which there would be no need to specify the host system for a given file. A second proposed feature is the "byte-stream" file, which is a pseudo-device which acts like a communications pipeline between two programs (resident on one TENEX system), having an input end on which one program writes and an output end from which the other program receives information. Our description below relies heavily on the JSYS trapping mechanism and the byte-stream file. The JSYS trapping mechanism has been programmed but is available only in an experimental version of the TENEX monitor. The byte-stream file mechanism has not yet been implemented, and recent indications are that it may not be available for some time due to higher priority TENEX work.

4. Mechanisms for Inter-NCJ Communication -- Byte-Stream Files and Channels

Let us look in a little more detail at the computational network. Each NCJ is a complete job on some TENEX site, and can thus consist of an entire fork structure of cooperating processes executing under the TENEX operating

system. These processes might include ones coded in INTERLISP, FORTRAN, BCPL, MACRO-10, etc. However, as seen from other NCJ's each NCJ is a unit with (possibly) several input connections, and (possibly) several output connections. While the functional characteristics of the input and output connections are important, there is no reason for one NCJ to know how the connections of another NCJ are associated with its different forks. Thus, a connection can be viewed as going from an output port on one NCJ to an input port on another NCJ. This suggests that each input(output) port on an NCJ be associated with a network socket on the associated TENEX, and that a communications connection between two NCJ's be simply an ARPANET connection between the two network sockets. The TENEX system makes it easy for a process to open a network connection as a file for input or output, given the host and socket numbers at both ends. Figure 1 shows a configuration of three NCJ's connected in such a manner.

There are two problems, however, in simply implementing inter-NCJ communication by means of network connections:

1) We wish to have the NCJ's precompiled, but the distribution of NCJ's to network sites should be determined on the basis of the load factors, etc., at the time the multi-host program is to be run. Thus, the compiled NCJ's could not contain fixed calls to the TENEX system to open network connections, since

although the socket numbers could conceivably be fixed, the host numbers would not be known when the NCJ is compiled.

2) In the process of debugging, and often in the process of watching the operation of a running multi-host program, it would be extremely valuable to be able to monitor the information flowing through the various inter-NCJ connections. It may be useful to have copies of the information fed into script files or monitoring programs, or to "dummy up" a malfunctioning NCJ by replacing its output with the contents of a file or with the user's own typed output. There is no convenient way to do this if the inter-NCJ connections are simply network files on the TENEX system.

To solve these problems we propose a facility we refer to as an inter-NCJ channel, which would be a named pipeline carrying a stream of bytes across the ARPANET from one (port on an) NCJ to (a port on) another NCJ. The NCJ's would be specified to the CNCP independently of the hosts on which they were to be run, and the channels could be rerouted, spliced into and multiplexed after the network of NCJ's was established. The proposed byte-stream file capability in TENEX would provide a useful set of primitive (and not so primitive) operations for the type of interjob and interprocess communication, synchronization and control

needed to implement these inter-NCJ channels.

One possible way of using the byte-stream file is to have all inter-NCJ channels mediated by such files. For each input(output) port an NCJ would open a corresponding named byte-stream file for input(output). The CN residing on the same host would open the same byte-stream file for output(input) and then create a fork which would copy all bytes from that file out over (in from) a network connection file which had been established by the CNCP to connect the NCJ's involved. Figure 5 gives the proposed actual implementation of the conceptual connections shown in Figure 1. The byte-stream files would provide a clean way to delay the binding of the named inter-NCJ channels with the actual network connections needed to implement them when the NCJ's were assigned to specific hosts. The names of the communications channels could be used as the names of the byte-stream files to be opened, and these could be compiled into the actual NCJ programs, without worrying on what host the NCJ or its communications partners were to be run. The ability to have byte-stream files opened for reading by more than one process, and the ability to splice into byte-stream files to provide intervening processing, would provide the basic tools needed to allow monitoring, production of script files, flexible reconnection, and other useful capabilities.

There is an inherent inefficiency in the approach

described above, since there are two user processes intervening between the communicating NCJ's: the process which copies bytes from the byte-stream to the ARPANET connection and the one which copies them back out (both are labelled COPY in Figures 4 and 5). It is conceivable that these COPY processes would have to be scheduled for each byte to be transmitted, with the obvious tremendous increase in overhead. Of course, the actual transmission over the network takes a noticeable amount of time, so it is unclear how bad this overhead would be.

Because the TENEX byte-stream file capability may not yet exist when the multi-host programming system is implemented, an independent version may have to be built. While doing this one could provide a hybrid notion which combines the byte-stream file and network file connection. The availability of a convenient mechanism to trap the JSYS monitor calls used for I/O will make the project much easier than might otherwise appear. Thus it should be possible to provide a flexible and reasonably efficient mechanism for implementing inter-NCJ communication.

5. Access to Non-Local Files

Many programs require access to previously established data files, and many write information on files to be shared

with other programs (or the user). Since the NCJ's are to be able to run on any of a number of hosts, they must be able to conveniently read and write on files which may be on other hosts, and the program must be able to do this independently of the host on which the NCJ is currently running. Luckily, the current version of the TENEX RSEXEC allows the user to maintain a "distributed file system" with files at several ARPANET TENEX sites, and to access them at RSEXEC command level without necessarily knowing where the files are. When the JSYS trapping mechanism is available, this distributed file system capability will be extended to the JSYS (essentially machine code) level. It will then be possible to run programs written in most existing programming systems on TENEX and still be able to have the programs access and write on the correct files independently of the host on which they are run. Of course, scratch files need not be accessed on or written on remote hosts. In any event, this problem seems to be one which will essentially disappear (as far as the multi-host robot system is concerned) as soon as the JSYS level distributed file system is implemented in the TENEX operating system.

6. Information Needed to Initiate a Multi-host Program

In order to initiate the computational network, the CNCP needs two types of information:

1) a description of the topology of the computational network (i.e. the inter-NCJ connections to be established) and a description of the structure of the NCJ's themselves.
2) ARPANET TENEX status information (e.g. load averages for the hosts on the ARPANET), preferential sites for the user, user names and passwords to be used, etc.

If we assume some sort of byte-stream file mechanism connecting the various NCJ's, then the first type of information -- a description of network topology -- can be relatively simple. One possible format might be a set of descriptors of the form:

CHANNEL = [RNCJ] RBSF <= [SNCJ] SBSF

where RNCJ and SNCJ are user names for two node computational jobs, and CHANNEL is the name used to refer to the inter-NCJ channel on which SNCJ sends information to RNCJ. RBSF is the name of the byte-stream file from which RNCJ expects to receive information, and SBSF is the

byte-stream file to which SNCJ sends information.(*)

Only enough of the structure of an NCJ must be communicated to the CP to permit it to initialize the fork structure of the NCJ. Given the current plans for implementing file transfers in a multi-host TENEX environment, there would be relatively little difficulty in providing such a description. It is merely necessary to give the name of a SAVE file for the top fork of the NCJ. Multi-fork structures are usually created by each fork doing a GET for each desired subordinate fork, so that the fork structure of a process (and hence an NCJ) is defined by the execution of GET's by the running program. The planned JSYS trapping mechanism for distributed TENEX operation will trap such GET's and determine where the desired SAVE file is located, and whether it must be moved to the local disk to be loaded.(**) or can be loaded directly from a network file. Thus, each CN can simply perform a GET on the top fork for its associated NCJ, establish the needed interhost network connections to the other CN's, and attach them to the proper ends of the desired byte-stream files.

---

* It is possible that there may be a need for other information about the desired characteristics of the communications link, such as the byte size, buffering characteristics, etc. Such information could be easily added to the channel descriptions above.

** Since shared files are PMAP'ed they must reside on the local disk.

Once all NCJ's have been loaded and all network connections and byte-stream files have been initialized, each CN can start to run the top fork of its associated NCJ. The top fork of the NCJ will bring in lower forks as needed, with the JSYS trap mechanism ensuring that the needed files will be found without the user program having to worry about their location. The various forks will connect to the other sides of the byte-stream files created by the CN, and the computational network will be established.

Obtaining the second type of information -- that relevant to the available TENEX sites on the network -- should not be difficult. ARPANET TENEX status information is already maintained and updated by the existing RSEXEC system in operation at the various TENEX sites. Information about site preference for the NCJ's involves several factors. The simplest and easiest to represent is the list of sites on which the user has accounts. This might be complicated depending on whether the user has varying financial resources or use restrictions at different sites. In addition to these essentially program-independent administrative constraints, there are constraints on the appropriate sites for each NCJ. The simplest, but least desirable, way of indicating such constraints would be to give a list of the acceptable sites (by name) for each NCJ, without giving any explicit (i.e. available to the CNCP)

description of what properties of the (computational equipment of the) sites made them acceptable for the given NCJ. It would be preferable to describe these constraints in terms of functional capabilities (e.g. available disk space, load, special I/O devices like an IMLAC, E&S display, plotter, XGP, etc.) since then the description of the total network job would be independent of the state of the network. That is, if new TENEX sites with different capabilities were added to the network, the CNCP would be able to make use of them without the user having to explicitly change his description of constraints on the sites for the NCJ's. (Of course, in either case there would be no conceptual problem if TENEX sites were removed, since the CNCP already has to deal with the fact of life that sites "officially" on the network may be temporarily unavailable due to machine or network malfunction or administrative fiat.)

There are still many questions to be resolved: What information is to be used in determining how the NCJ's are to be distributed? What are the criteria to be met (e.g. least cost, greatest speed, least load on particular resources, etc.) and what algorithm is to be used to determine how to meet these criteria? How is the user to provide the necessary information (convenient formats)? Is it better to have the information for the constraints for each NCJ stored in association with the NCJ, or to have a

single file describing both topology and node constraints? Should the user-dependent information (i.e. sites, accounts, passwords, etc.) be stored separately from the description of the particular computational network? How is the CNCP to access both user-supplied and ARPANET-supplied information? How are sensitive pieces of information (such as passwords) to be stored in the files which describe the computational network?

7. Controlling, Communicating With, and Debugging Multi-Host Jobs

Once the computational network has been set up and the NCJ's are running, there still remains the task of controlling such a distributed job, and the more difficult task of debugging it when it goes wrong. There is unfortunately very little to go on in designing this part of the system, so that the following ideas are merely a first attempt to sketch the control, communications, and debugging facilities which might be useful.

For communication, at the first level, the problem is basically a simple one. The user has just one terminal, but there are many possible sources of output and many places the user might want to direct terminal input. For each NCJ there are the primary input and output files for each fork

in the job (though this usually reduces to a single teletype input and output). Then, the user may wish to monitor traffic on the various inter-NCJ input and output channels. There is also a pair of inputs and outputs for the CN which controls the given NCJ. In addition, it is possible that the user may need the capability of making direct contact with the EXEC level of the TENEX system at each of the sites, acting as the controlling teletype for each NCJ. Thus, we have a situation in which the user may need to use a single terminal for at least three input streams and three output streams for each NCJ. The possibilities for confusion and error are obvious.

Controlling the input streams is probably the simplest problem. It is merely necessary to give the user some means of breaking back to the main CNCP input and then indicating to which input file he wishes to connect his terminal. The only system design problem seems to be to tread the thin line between making the switching from one file to another require so long a protocol that it is unusable, or making the protocol so short and free of redundancy that no error checking is possible. In the latter case it is likely that garbage will be sent to many input files due to mistyping or terminal errors. Another difficulty may arise if the user is given the capability of typing ahead on one or more input files. If part of the multi-host system goes bad, it may be necessary to flush the type-ahead in various input streams,

and it is not clear whether this should be automatic (under what conditions) or left to the user (and hence sometimes neglected or performed too late).

On the output side the situation is more muddy, since there may well be several output channels contending to be printed at once. Of course, they cannot just be intermingled on a letter-by-letter basis, for this will produce garbage. It might seem more reasonable to have only one output channel connected to the terminal at once, but that leads to the difficulty that a vital message on one channel may never be seen, or may be seen too late. One could connect several channels, allowing each one to type a full line (starting with a channel-identifier) before switching to another channel. This is open to the problem of one channel never finishing a line, and thus locking out other channels.

We have no solution to the output problem currently. A tentative suggestion for a basic set of capabilities for controlling output are the following (modified from the current TELNET system): The user could declare output channels to be in three categories. The first category, the "active" channels, would be routed to the terminal, with channel-identifier whenever they transmitted output (possibly with a provision for a single active channel to continue printing out until the end of a line or some time

limit has been reached). The second category, "high-priority" dormant channels, would have their output queued, and some sort of signal would be sent to the user indicating that there was data on the channel. The last category would merely be queued with no signal, for the user to interrogate at his convenience. A possibly useful modification to this scheme would be to permit the specification as to "active", "high-priority dormant" and "low-priority" to be made on a message-by-message basis, with the transmitting program sending a header which specifies the classification it requests for the current message.

In addition to the problems of terminal I/O, the CNCP should provide other facilities for controlling the information flow among the various NCJ's. It would be useful to send copies of the information flowing along one or more inter-NCJ channels to one or more receiving processes. This would make possible the creation of "typescript" files for documentation or debugging, and would facilitate the insertion of various monitoring facilities which would be useful in debugging. When the information flowing along a channel is directed to one of these alternative processes, it should be possible to specify if the information is to continue to flow to the original destination. A further possibility would be to allow the insertion of a process between the sender and receiver, to

act as a "translator".

In the area of control primitives for allowing debugging of multi-host programs, we are clearly in the dark. Some possible ideas are:

a) Allow the user to splice in a local debugger (e.g. IDDT) at each NCJ

b) Permit the user to specify breakpoints in one or more NCJ's, and allow each breakpoint to suspend the operation of a specified subset of the NCJ's, not merely the one in which the breakpoint occurred

c) Allow the user to insert monitoring programs (see above) in various inter-NCJ channels, and to halt a subset of the NCJ's (as in b) when the information in a given link meets some condition

d) Permit the user, or a process specified by him, to be used as the input to a specified inter-NCJ link, to substitute for a possibly malfunctioning NCJ.

In addition, it would seem useful if the directives given to the debugger, and in particular the descriptions of the conditions under which a break is to occur and the operations to be performed at the break, were to be specifiable in some convenient higher level language. This is already the case for INTERLISP, but is certainly not true for the current DDT package. This feature would be quite useful in debugging current FORTRAN and MACRO programs, and

would probably be of even greater use in the more complicated environment of a multi-host system.

8. Final Remarks

We have attempted above to record some of our thoughts on the design of a system for simplifying the construction, debugging, and operation of multi-host programming systems. Some of the facilities proposed for such a system are due to the particular properties of the ARPANET. For example, the exact structure of inter-host communications protocol and file-transfer protocol has affected the design of the inter-NCP channels. Some of the facilities would be needed for any system of several programs interacting by means of communications links, rather than by shared core or flags in an operating system (e.g. a facility for monitoring the inter-program communications channels). Other facilities would be useful in the context of controlling and debugging several simultaneously active, mutually communicating programs (e.g. communications facilities for the user, debugging facilities able to control and monitor the execution and communication of several simultaneously active processes).

C. Miscellaneous Assistance with the JPL Robot Project

The assistance we provided during this quarter to the JPL Robot Project included the following:

1) The BBN ARPA network group consulted with the JPL staff on the many hardware and software issues associated with interfacing the JPL machines to the ARPA network

2) The JPL IMLAC group was given assistance in putting up and running the BBN robot world display software

3) TENEX computer time and file storage space were provided to various members of the JPL staff as back-up when their other computer services were unable to meet their needs.